Essay on Semantics Definition in MDE. An Instrumented Approach for Model Verification

Benoit Combemale, Xavier Crégut, Pierre-Loïc Garoche, Xavier Thirioux

Essay on Semantics Definition in MDE
An Instrumented Approach for Model Verification

Benoît Combemale[⋆] Xavier Crégut[†] Pierre-Loïc Garoche[‡] Xavier Thirioux[†]

[⋆] *INRIA*, Ecole des Mines de Nantes, France
[†] *Institut de Recherche en Informatique de Toulouse*, Université de Toulouse, France
[‡] *Office National d'Étude et de Recherche en Aérospatiale*, Toulouse, France

first_name.last_name [⋆]: @inria.fr [†]: @enseeiht.fr [‡]: @onera.fr

Abstract

*In the context of MDE (*Model-Driven Engineering*), our objective is to define the semantics for a given DSL (*Domain Specific Language*) either to simulate its models or to check properties on them using model-checking techniques. In both cases, the purpose is to formalize the DSL semantics as it is known by the DSL designer but often in an informal way. After several experiments to define operational semantics on the one hand, and translational semantics on the other hand, we discuss both approaches and we specify in which cases these semantics seem to be judicious. As a second step, we introduce a pragmatic and instrumented approach to define a translational semantics and to validate it against a reference operational semantics expressed by the DSL designer. We apply this approach to the* xSPEM *process description language in order to verify process models.*

1 Introduction

In the MDE (*Model-Driven Engineering*), models are defined by means of metamodels which specify their syntax and give some structural properties that constrain valid models. Usually they rely on an abstract syntax (specified by means of metamodeling languages like MOF [42], Ecore [11, 10], KM3 [27] or others) enriched with constraints expressed using query languages like OCL [41]. The MDE practices eases as well the definition of DSL (*Domain Specific Language*). These languages allow users to concentrate on their problems because they manipulate a formalism specific to their activity. Numerous available frameworks (Topcased [22], GME [34], AMMA [6], etc.) have emerged, allowing to easily define both concrete and abstract syntax of such DSLs.

A current open issue is the expression of a behavioral semantics allowing to execute models during the development process. Our actual works in this context has its roots in the different kinds of behavioral semantics devised for programming languages engineering (e.g., [56]).

We carried out several experiments to define an operational semantics for a simplified process modeling language. We have used metaprogramming languages (like Kermeta [38]) and endogenous transformations (expressed in ATL [28], or in AGG [53], a rewriting graph tool). We then compared these different approaches in [18]. In both cases, we were able to execute a model but a mandatory preliminary step was to extend the metamodel in order to describe the additional pieces of informations required to capture a snapshot of the system. Different approaches may be followed depending upon the kind of extension. A metaprogramming approach requires to enrich the metamodel with implemented operations. Instead, these operations are expressed in an endogenous transformation approach, through the transformation itself.

A second experiment was the definition of a translational semantics to Petri nets using ATL exogenous transformations[1]. The obtained Petri nets could then be executed using their own semantics. This translation defines an behavioral semantics for the original DSL. Once the semantics is defined by translation to another language, we were able to reuse existing tools such as model-checkers or simulators available on the target model. This approach seems powerful but one of its main drawback is the interpretation of tools results back on the source model.

A lot of works consider this concern of defining a semantics on a DSL. Our proposal is mainly focused on the

[1] see. `http://www.eclipse.org/m2m/atl/usecases/SimplePDL2Tina/`

way used to express the semantics but also on means to validate its consistency with respect to the one intended by the DSL users. We do not target a general solution for every kind of semantics and for every DSL but we rather propose a methodology that could be instantiated in many contexts depending on the DSL, on the granularity of the needed semantics, and so on.

Thus, this paper broadens our tries and proposes:

- A survey (Section 2) synthesizing the ways of formalizing the semantics definition in the model-driven development context. It particularly addresses the two principal approaches: exhibiting an operational semantics, mainly through an endogeneous transformation (based on rewriting rules, automaton, etc.), or a translational semantics which relies on a separate formal model to embed the model semantics.

- A definition (Section 3) and a complete use (Section 4) of our proposal for the definition of a behavioral semantics. It starts with the definition of the reference semantics of the initial metamodel, goes on with the definition of a translational semantics, and ends with the proof of bisimulation stating that both semantics – the reference one and the one obtained through the translation – characterize the same systems[2].

This second contribution presents a pragmatic approach to define a translational semantics and reuse existing tools of the chosen target domain. It is illustrated with an example that considers process models and embeds them into prioritized time Petri nets. Associated tools, such as the Tina model checker, can then be used to observe properties of the models. Finally, we detail a bisimulation proof that validates the semantics defined by translation against the reference one defined on the source DSL.

The paper is organized as follows. Section 2 presents a survey of ways to define the semantics of a DSL and discusses benefits and weaknesses of operational and translational semantics. Section 3 describes the general approach we propose to define the semantics of a DSL and ensuring their consistency. It is illustrated in Section 4. Last section gives some concluding remarks and future works.

2 Defining an Execution Semantics for DSL

2.1 Taxonomy

The intensive works on the semantics of programming languages have provided a taxonomy of the different techniques used to express a semantic according to different needs [56]. This concern is much more recent for modeling languages. We can identify three main techniques to define the behavioral semantics of a DSL.

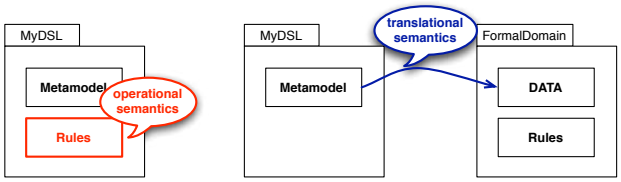

Figure 1: Operational Vs. Translational Semantics

The first one is called *axiomatic semantics*. It consists to define a set of properties satisfied by the model at the different steps of its execution (like pre- and post-conditions). Unfortunately, it usually not easy to fully specify the behavior of the model [56]. Furthermore, an axiomatic semantics can not be made automatically or easily executable.

Operational semantics is the second technique. It directly manipulates the model. Thus, it allows to stay in the same technical space and to express the evolution of the model state in the same specific domain (fig. 1, on the left). It generally requires to extend the initial metamodel with the informations that describes the state of model at execution. Several possibilities have been explored to implement the operational semantics directly on the abstract syntax.

The first one is to use a metaprogramming language to express directly the behavioral semantics like a set of operations for each concept. We can cite, for example, operations defined with Kermeta [38], xOCL [16] or the MOF action langage [9, 45].

The second way is to lay endogenous transformations over the abstract syntax. They can be implemented using any model transformation language.

As an example, [35] uses QVT [44] to express in-place rewriting rules that gradually compute the values of an OCL [41] expression. In this way, they have defined an operational semantics of OCL and are able to compute the value on an OCL expression. The authors had first to complete the OCL and UML metamodels to add the required missing dynamic informations.

Endogenous transformations have also been widely implemented through graph transformation [51]. Graph transformation provides a declarative and rule-based technique to define an operational semantics, but also analysis capabilities due to its formal nature. AGG [53] is an example of such a language that is directly usable over Eclipse/EMF models thanks to [7]. As another example, the GROOVE tool was used in [29] (and detailled in [30]). Kuske, Gogolla et Ziemann [32, 33, 24, 57] are also using graph transformations (and the notion of *transformation unit* [31]) to define

[2] In this paper, we follow the definition of bisimulation given in [37]

the behavioral semantics for some UML diagrams and their relationships.

Hausmann [25] introduces the notion of dynamic metamodeling (DMM) as a semantics description technique for Visual Modeling Languages. Graph transformation is used to define the behavior as a system of transitions. In [21], graph transformation rules are visually defined thanks to collaboration diagrams. [23] represents elementary transformations as UML collaborations diagrams indicating the elements to add and/or to remove when it is applied. These transformations are embedded in the state of a UML activity diagram that controls the order the transformations are applied. It thus looks like a graphical meta-programming language (called *Story Diagrams*) where actions are transformations based on graph rewriting and control structures are provided by the UML activity diagram.

The third technique to define the behavioral semantics of a DSL is called *translational semantics*. On the contrary of operational semantics, a translational semantics maps the model state into another (formally well defined) technical space (fig. 1, on the right). Thus, it relies on an existing semantics defined on the target technical space. It consists to translate constructs from the initial domain into the constructs of the formal target space. That is this translation that gives the semantics of the initial domain.

As part of the MIC approach (Model-Integrated Computing), the ISIS laboratory promotes semantics anchoring [12] that is a kind of translational semantics. It consists to map the DSL constructs into a semantics unit to define its semantics. The GReAT transformation language is used [1]. We can notice that semantics units are defined using operational semantics, for example using Abstract State Machines (ASM). Translational semantics is also used by the group pUML[3], called *Denotational Meta Modeling*, in order to formalize some UML diagrams [13].

Numerous works use translational semantics, mainly to take advantage of the facilities and tools available in the target technical space (code generators, model-checkers, simulators, visualization tools, etc.). To deal with the complexity of a translational semantics definition and help in handling changes in the language definition, Cleenewerck *et al.* [17] promote the separation of concerns. They define a *language module* as a language construct accompanied by its translational semantics that constitutes an important design decision in the language. The constructions (i.e., the concerns) can then be assembled in order to define the semantics of the entire DSL.

Other taxonomies have been proposed. Clark *et al.* [14], recently updated in [15], share the distinction between operational and translational semantics. Their works are focused on the execution semantics and therefore do not mention axiomatic semantics. They also define the notion of semantics by extension, consisting in extending the concepts and semantics of an existing language and thus allowing for capitalization and reuse of semantics. Nevertheless, the semantics is defined either as a operational or translational semantics. Finally, we do not share the definition of a denotational semantics as a mapping to a semantic domain. For us, this is the general definition of a semantics and thus generalizes all the other kinds.

Hausmann [25] also presents a taxonomy of techniques to express a behavioral semantics. He lists the available techniques to achieve specific objectives (e.g., verification of properties, analysis of the consistency and code generation) and he identifies the general techniques to express the semantics, including operational and translational semantics.

2.2 Discussion

An operational semantics seems simpler to define and to use than a translational one because it is directly expressed on the concepts of the specific domain which are naturally well-known by the expert. For the purpose of animation (viewing the evolution of a model during its execution) and/or simulation (analysing an execution) this approach seems preferable, in particular if the model of computation is fairly simple, for example representable using discrete states.

However, operational semantics may not always be easy to implement. For example, if the model of computation deals with time, operational semantics definition may become tricky and may involve to heavily extend the source metamodel to deal with time constraints [48].

Furthermore, if one needs to use formal techniques like model-checking for example, a translational semantics seems more relevant than operational one. Indeed, state-of-the-art existing tools rely on several years of research and development and could not be easily generalized to be applied to any domain specific concepts.

To use translational semantics, one has to choose the appropriate target technical space depending on the kind of property one wants to check or depending on the tool one wants to use. This approach requires to define a metamodel for the target language, which may not already exist in a MDE model flavor, and then to define a translation from models of the source language to models of the target one. Finally a concrete syntax extractor is needed in order to create the input data for the tools.

So one great benefit of the translational semantics is to give access to any tools existing on the target space. Obviously, it requires a good understanding of both the source space, specific to a given domain, and the target one, gen-

[3] *The precise UML group*, cf. http://www.cs.york.ac.uk/puml/

erally a more formal one. Indeed, the execution of a model in the source language must be expressed by a translation to the target language, relying on its behavioral semantics. Another difficulty of the translational semantics is that results obtained in the target space have to be interpreted back according to the concepts of source one.

The work done in [49, 50] illustrates the strength and weakness of operational and translation approach explained here-above. The authors aim at formally defining the semantics of a DSL by translating it to the Maude formal environment. The Maude tool is then used to express the operational semantics using rewriting rules. It looks like semantics anchoring approach presented in the previous section with ASM replaced by Maude that the purpose is not to define reusable semantics units but only express the desired semantics. An identified drawback is that it requires specialized knowledge and expertise by the DSL designer who has to use the Maude tool. Thereafter, the same authors have proposed in [47] the use of graph transformation to directly express the operational semantics through the abstract syntax of the DSL, which then become more intuitive to the designer. Metamodel and graph transformation rules are then automatically translated into the Maude environment.

3 A Pragmatic and Combined Approach to Define Consistent Behavioral Semantics

We introduce in this section our pragmatic and combined approach to define a behavioral semantics for DSL. We first discuss the necessity to define a reference operational semantics ¡refletant¿ the DSL designers experiences. Then we discuss the validation of translational semantics according to the reference one and its abstraction level.

3.1 The Need for a Reference Semantics

Having in mind the variety of choices that could be made in order to express one DSL's semantics, the first concern should be to define a reference semantics, which should carry the most precise (or less abstract) view on model executions. So we have to gather and take into account every concept that every expert has pinpointed as important according to his own standpoint. For instance, some expert may stay focused on complex functional properties, while some other would direct his interest toward real-time aspects only. Obviously, independent concerns should better be expressed independently. As shown in section 4.3, we have defined a translational semantics which indeed respects such a separation of concerns. In our opinion, a reference semantics should stay as close as possible to the model designers' views, and as such, should also stick to the original technical space, avoiding its translation to a somehow distant target space. As a consequence, a reference semantics ought to be operational, rather than translational or axiomatic.

More than being only the most precise one, the reference semantics also gives us the opportunity to define a formal semantics, i.e., one that could serve as a basis for formally expressing a specification and achieving mechanized validation proofs. Model designers may also consider other semantics, which may be designed for practical purposes, as long as these semantics are respectful of the reference semantics. In that case, designers will also benefit from the specification and proof effort made on the reference semantics, as some properties will automatically be carried over to these other respectful semantics. The forthcoming problem of determining whether another given semantics is respectful of the reference semantics will be addressed by exhibiting simulation or bisimulation relations between these semantics.

The bisimulation relation between a reference semantics and any other translational semantics is defined as follows, as pictured in Figure 2. Note that it is a kind of weak bisimulation, assuming the reference semantics has no τ unobservable transitions. This hypothesis makes sense as transitions introduced by model designers are always meaningful and reflect observable changes in the model.

Definition 1 (Weak Bisimulation Relation) *Let us assume a translation function* Π *between state spaces from a reference semantics* (RS) *to a target semantics* (TS)*: For all model state* $S \in RS$ *and for all sequence* $u \in T^*$ *such that* $S_0 \xrightarrow{u} S$*,* S_0 *being an initial state of RS:*

1. $\forall \lambda \in T, S' \in RS,$
$$S \xrightarrow{\lambda} S' \implies \Pi(S) \xrightarrow{\tau*} \xrightarrow{\lambda} \xrightarrow{\tau*} \Pi(S')$$
2. $\forall \lambda \in T, P \in TS,$
$$\Pi(S) \xrightarrow{\tau*} \xrightarrow{\lambda} \xrightarrow{\tau*} P \implies \exists S' \in RS \; s.t. \begin{cases} S \xrightarrow{\lambda} S' \\ \Pi(S') \equiv P \end{cases}$$

where $\xrightarrow{\tau}$ *denotes a non observable transition.*

We propose in the remainder a formal operational reference semantics, as a rather standard transition system. As is, this transition system may be directly implemented by a set of endogenous rewrite rules, mapping each state to its possible successor states. This approach can be implemented by using a model transformation engine such as ATL [28] or a graph transformation tool such as AGG [53].

3.2 Taxonomy of Combined Semantics Definition

In this section, we enumerate the different approaches used to describe a model semantics relying on a target

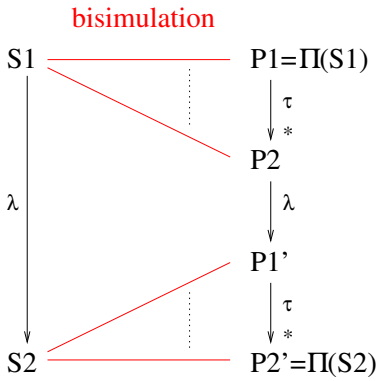

Figure 2: Bisimulation between a reference and any other semantics

model with its own well defined semantics. In a general manner, we consider in the following that the translation target model is provided with a formal small-steps operational semantics, like the one of Petri nets. The source DSL semantics can then be described by two orthogonal viewpoints: the abstraction level and the kind of description. Depending on the abstraction level used to describe the source DSL semantics, we then identify how to ensure the quality of the defined translational semantics.

3.2.1 Expressing the source DSL semantics

A first characteristic of the source DSL semantics is its **abstraction level**. A small-steps operational semantics could, for example, be described by a set of rewriting rules, by an automaton or a Kripke structure. A denotational abstraction of this semantics maps each *observable* state to its possible observable images by one or more applications of these transitions or rules. Finally an axiomatic semantics, abstracting the latter, is not operational but rather defines a set of properties satisfied by the model at the different steps of its execution (like pre- and post-conditions). It does not fully specify the behavior of the model [56].

The second, very pragmatic, viewpoint is the **kind of description** of this source semantics. Independently of its abstraction level, the semantics can be described more or less formally. Historically, DSL were used by designers to communicate their modeling concepts. Therefore there is no general formal framework for specifying the DSL semantics. Thus the semantics description, when it exists, is usually given informally in natural languages. Sometimes it is defined using more formal structures such as Kripke structures, rewriting rules or endogenous transformations. If the semantics does not explicitly exist, whether in a formal manner or not, its definition is a required step.

We now consider the different possible combinations of these two DSL semantic characteristics in order to identify the key steps during the definition of a sound translational semantics. Whatever the precision of the initial semantics, a domain expert identifies equivalent states with respect to the properties of interest for the model. Each such class is characterized by a state predicate (predicate abstraction phase). An event or a state evolution in the model is said to be observable if its states before and after the event are not equivalent. In the following, the translation function considers a model in a particular state. The image of such a state by this function is, in the following, a Petri net with a particular marking.

3.2.2 Translation from an Axiomatic Semantics: Expression of the Consistency

When the initial semantics is not precise enough, it is not always possible to exhibit a one to one mapping between this semantics and the translational one. However, we need to be sure that the target model satisfies the properties expressed in the initial semantics. A standard axiomatic semantics contains invariants, preconditions and postconditions which must be expressed according to state predicates defined by the expert. On a more theoretical side, there is no bisimulation but simply a simulation of the target model by the source model. In fact, the axiomatic semantics is not operational, but the properties of the axiomatic semantics permits to define a set of observable state-based properties on the target model, that will have to be checked. We will be able to translate only types of properties supported in target technological space, in our case behavioral properties of the Petri nets. The possibility of translating axiomatic properties is thus strongly dependent on the target DSL.

This approach gives to model designers a way to ensure minimal requirements such as typing constraints, but light axiomatic semantics does not give strong confidence in the results of target model analysis tools. Actually, the definition of a semantics by translation requires to make a lot of choices in the semantics definition and the resulting semantics could be distant from the original one, while being compatible with the axiomatic requirements.

3.2.3 Translation from an Operational Semantics: Bisimulation Relation

The approach of translating from an operational semantics is more promising as the distance between the original and target semantics is smaller.

When semantics is formally expressed and described in an operational way, one has to ensure that the original semantics and the one obtained by translation describe the same behaviors. Thus one needs to have a proof of bisimulation between these two semantics. It is a proof by induction on the abstract syntax in which one shows that any transition in the first semantics corresponds to a transition

in the second one, and vice versa. If one of these semantics comprises more states than the other, it is a weak bisimulation. This proof guarantees that the observable events of these two semantics are identical and thus that the analyses on the target model are relevant for the source model properties. The bisimulation consists in showing that a model and its image by the translation function have identical observable events at every moment.

In the next section we propose such a model definition, as a translation from an operational semantics defined on process models to Petri nets. This translational semantics definition is then validated by the weak bisimulation we exhibit in the section 4.5. This application ensures that we can rely on existing tools for Petri nets to verify properties on process models.

3.3 Our Approach in a Nutshell

Whatever be the kind of description of the reference semantics, the general schema of our proposal is described by the following steps for a given metamodel:

1. reference semantics must be defined according to the DSL designers needs.
2. an adequate target domain must be chosen depending on the intended facilities;
3. then a mapping from the initial DSL to the target domain must be defined;
4. a validation of the mapping must be established with respect to the reference semantics thanks to a bisimulation proof;
5. finally, any user of the initial DSL is able to rely on tools available in the target domain to observe properties of its initial models.

The last four steps (2–5) can be repeated to reuse facilities provided by different domains with possibly different levels of abstraction. It thus leads to the definition of a family of semantics.

4 Process Models Verification Through Prioritized Time Petri Nets

We now illustrate the above methodology throughout for the instrumentation of a process model DSL: we enrich it with a semantics definition "by translation" with respect to its reference semantics. We first present our source DSL: xSPEM, a SPEM-based[4] process metamodel enriched with

[4]SPEM is an OMG specification [43]. It stands for Software Process Engineering Metamodel. SPEM is used to define software and systems development processes and their components. SPEM is a MOF-based metamodel [42].

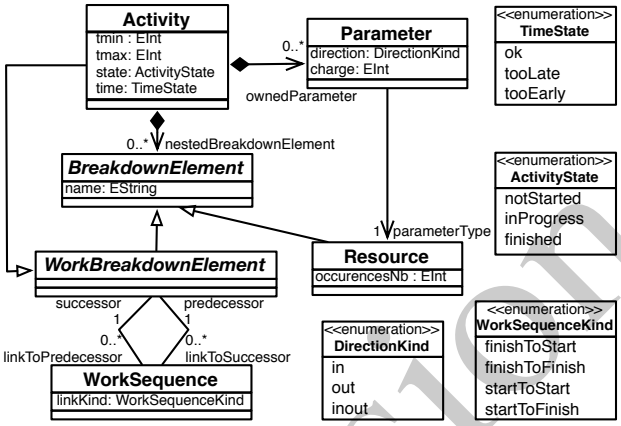

Figure 3: xSPEM metamodel (simplified)

information to support the execution of its models. We also explain how the DSL designer may define an abstraction of xSPEM operation semantics suited to the properties he wants to focus on. The second part presents prioritized time Petri nets and their associated semantics. Thereafter, we propose the mapping from xSPEM to Petri nets and express it in a MDE approach. Finally, we validate the semantics induced by the mapping with respect to the identified reference semantics. The full details concerning the bisimulation proof are given in Appendix A.

4.1 xSPEM: an eXecutable SPEM metamodel

In our experiments, we used a simple process description language, the simplified xSPEM metamodel (cf. Figure 3).

xSPEM stands for *eXecutable SPEM*. It is proposed in [2] as an extension of SPEM2.0 specification [43] in order to take into account the support of process enactment while remaining standard. In the metamodel, an *Activity* represents a general unit of work assignable to specific performers. It may rely on inputs and produces outputs (represented by *Resource*). An activity may be broken down into sub-activities. Activities are ordered thanks to the *WorkSequence* concept whose attribute *linkKind* indicates when an activity can be started or finished. The values of *linkKind* are defined by the *WorkSequenceKind* enumeration. One value is named in the form *state*To*Action* where *state* indicates the state that must have been reached by the source activity in order to perform the *action* on the target activity. For example, linking two activities A_1 and A_2 by a *WorkSequence* relation of kind *finishToStart* specifies that A_2 will be able to start only when A_1 is finished. The *direction* attributes defined in *Parameter* could be used to complete sequencing constraints expressed through the *WorkSequence* concept.

In order to tailor a process model for a given project, additional features have to be defined. They are required to specify the number of used resources, expected duration, etc., and to identify the concrete resources allocated to the project.

xSPEM includes: 1) the time interval during which an activity must finish once started (*tmin* and *tmax* on *Activity*); 2) the number of occurrences for one kind of Resource affected to the project (*occurrencesNb* on *Resource*); 3) the work load affected to a resource for an activity (*charge* on *Parameter*).

In order to enact a process model, its semantics has to be defined or at least validated by the DSL designer. So we consider that we should not yet rely on a translational semantics but on an operational semantics that explains how a model/program of the DSL evolves. It thus consists in identifying model states and transitions between these states. As the initial definition of the DSL mainly focuses on static properties of the domain language, a firt step consists in adding features to the metamodel to capture states.

xSPEM identifies two orthogonal aspects for the *Activity* element. First, an activity can be *not started*, *inProgress*, or *finished* (*state* attribute). Secondly, there is a notion of time and clock associated to each activity; but this time is only relevant for transition-enabling conditions (in our case transitions that start and finish an activity) and is not explicit in state properties. Thus it is abstracted away and yields the finite set of observable states {*tooEarly*, *ok* , *tooLate*} (*time* attribute). This second orthogonal aspect is only relevant when the activity is finished. Abstracting away internal clocks doesn't mean they are thrown away, but only that their values are not observable. In particular they will be needed and used in the bisimulation proofs.

It is also necessary to take into account the concept of a global external clock ($clock \in \mathbb{R}^+$), whose rate is followed by the internal activity clocks when performing idle time-elapsing transitions.

Definition 2 (xSPEM Model State) *We denote a state of an Activity by a triple* $(state, inTime, clock) \in \{notStarted, inProgress, finished\} \times \{tooEarly, ok, tooLate\} \times \mathbb{R}^+$. *A model state is then a mapping from each Activity to its state in the concerned model.*

xSPEM reference semantics An observational abstraction of the operational semantics of our processes with respect to our properties can now be defined. The expert has again to formalize the initial state and the transition relation. In our case, it is quite natural: the initial state is $\{a \mapsto (notStarted,\ ok, 0) | a \in \mathcal{A}\}$. The transition relation is defined for *Activity* in Figure 4. It is composed of two possible transitions: a first one allows to start the activity and the second one to finish it. An activity can be started whenever it is not yet activated and when its predecessor constraints are satisfied. Concerning the second transition rule, it has three cases depending on the value of the clock and the timing bounds of the considered activity.

4.2 Prioritized Time Petri Net

As xSPEM models real-time constraints and clocks, we find it natural to use time extensions to basic Petri nets.

Time Petri Nets (TPN) [36] are one of the most widely used model for the specification and verification of real-time systems. Time Petri nets are Petri nets (PN) in which a non-negative real interval $I_s(t)$, with rational end-points, is associated with each transition t of the net. Function I_s is called the *Static interval* function.

$\mathbf{R}^+$ and $\mathbf{Q}^+$ are the sets of non negative real numbers and rationals, respectively. Let $\mathbf{I}^+$ be the set of non empty real intervals with non negative rational end-points. For $i \in \mathbf{I}^+$, $\downarrow i$ denotes its left end-point, and $\uparrow i$ its right end-point (if i bounded) or ∞. For any $\theta \in \mathbf{R}^+$, $i \dot{-} \theta$ denotes the interval $\{x - \theta | x \in i \wedge x \geq \theta\}$.

Prioritized Time Petri Nets [4] extend TPNs with the priority relation $\succ$ on transitions. Priorities are represented by directed arcs between transitions, the source transition having a higher priority.

Definition 3 (Prioritized Time Petri Net – PrTPN) *A* Prioritized Time Petri net *(or* PrTPN*) is a tuple* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, \succ, m_0, I_s \rangle$, *in which* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ *is a Petri net,* $I_s : T \rightarrow \mathbf{I}^+$ *is a function called the* Static Interval *function and* $\succ$ *a pre-order on transitions.*

P is the set of *places*, T is the set of *transitions*, $\mathbf{Pre}$, $\mathbf{Post} : T \rightarrow P \rightarrow \mathbf{N}^+$ are the *precondition* and *post-condition* functions, $m^0 : P \rightarrow \mathbf{N}^+$ is the *initial marking*. Time Petri nets add to Petri nets the *static interval function* I_s, that associates a temporal interval $I_s(t) \in \mathbf{I}^+$ with every transition of the net. $Eft_s(t) = \downarrow I_s(t)$ and $Lft_s(t) = \uparrow I_s(t)$ are called the *static earliest firing time* and *static latest firing time* of t, respectively.

A Prioritized Time Petri net is given in Figure 5. In such example, the place $p0$ has a unique token. Both transitions t and t' could then be fired. However they have to satisfy both the timing constraint and the priority expressed between them. Here, at time 0, only transition t can be fired. But, in the valid timing range for both, *i.e.* in $]1, \omega[$, t' must be fired before t. In any case, when a transition is fired, it consumes a specified number of tokens and produces also a given number of tokens. These values are defined on input and output arcs of transitions. Default values for both are one token, *i.e.* a transition needs one token in the source place and produces one new in the target place. In this example, the transition t' consumes one from $p0$ and produces two tokens into $p2$. In our graphical syntax, the number of

Let a be a given activity.

$$\forall ws \in a.linkToPredecessor,$$
$$(ws.linkType = startToStart \;\&\&\; ws.predecessor.state = \{inProgress, finished\})$$
$$||\; (ws.linkType = finishedToStart \;\&\&\; ws.predecessor.state = finished)$$
$$(notStarted, ok, clock) \xrightarrow{StartActivity} (inProgress, ok, 0)$$

$$\forall ws \in a.linkToPredecessor,$$
$$(ws.linkType = startToFinished \;\&\&\; ws.predecessor.state \in \{inProgress, finished\})$$
$$||\; (ws.linkType = finishedToFinished \;\&\&\; ws.predecessor.state = finished)$$
$$(inProgress, ok, clock) \xrightarrow{FinishActivity} (finished, tooEarly, clock) \quad \text{if } clock < tmin$$
$$(inProgress, ok, clock) \xrightarrow{FinishActivity} (finished, ok, clock) \quad \text{if } clock \in [tmin, tmax[$$
$$(inProgress, ok, clock) \xrightarrow{FinishActivity} (finished, tooLate, clock) \quad \text{if } clock \geq tmax$$

Figure 4: Event-based Transition Relation for Activities

tokens in a place is specified by a number, when greater than one, or a black dot, when equal to one.

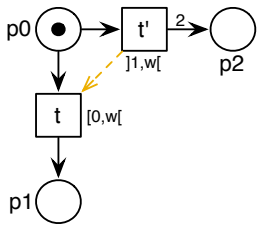

Figure 5: A Prioritized Time Petri Net

States, and the temporal state transition relation $\xrightarrow{t@\theta}$, are defined as follows:

Definition 4 (PrTPN state and semantics) *A* state *of a PrTPN is a pair $s = (m, I)$ in which m is a marking and I is a function called the* interval *function. Function $I : T \to \mathbf{I}^+$ associates a temporal interval with every transition enabled at m.*
We write $(m, I) \xrightarrow{t@\theta} (m', I')$ iff $\theta \in \mathbf{R}^+$ and:

1. $m \geq \mathbf{Pre}(t) \;\wedge\; \theta \geq \downarrow I(t)$
   $\wedge\, (\forall k \in T)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k))$
   $\wedge\, (\forall k \in T)(m \geq \mathbf{Pre}(k) \wedge \theta \geq \downarrow I(k) \Rightarrow$
   $\neg k \succ t)$
2. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
3. $(\forall k \in T)(m' \geq \mathbf{Pre}(k) \Rightarrow$
   $I'(k) =$ **if** $k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$
   then $I(k) \dot{-} \theta$ **else** $I_s(k))$

Transitions may fire at any time in their temporal intervals, so states typically admit an infinite number of successor states. As with many formal models for realtime systems, state spaces of PrTPN are typically infinite. Model checking PrTPN first requires to produce finite abstractions of their state spaces, that is labeled transition systems that preserve some classes of properties of the PrTPN state space.

Different state equivalence class constructions have been proposed and are available in TINA, preserving different families of properties of the state space. *State class graph* construction preserves markings of the PrTPN and all the properties that can be expressed in linear time temporal logics like *LTL*.

4.3 XSPEM2PETRINET Transformation

We now propose to implement the semantics defined in the previous section in order to check XSPEM process models. For this purpose, we formalize a transformation from XSPEM to Petri nets, thus defining a translational semantics.

The Figure 6 presents the mapping in a graphical view. The transformation is first defined by a structural mapping from a XSPEM model to Petri net without any marking. Then a second step is performed that produced a marking in the Petri net structure. This second step is defined by the value of the extra variables denoting the semantics state of the XSPEM model.

Structural Mapping Each *Activity* is translated into three places characterizing its state (*NotStarted*, *InProgress* and *Finished*). An additional place called *Started* is added to records that the activity has been started (and may either be *inProgress* or *finished*). It corresponds to the set identified in the operational reference semantics (cf. Figure 4). Three places define a local clock that may be in state *TooEarly* when the activity ends before *tmin*, in the state *TooLate* when the activity ends after *tmax*, and in state *ok* when still on time. Four transitions between these seven places define

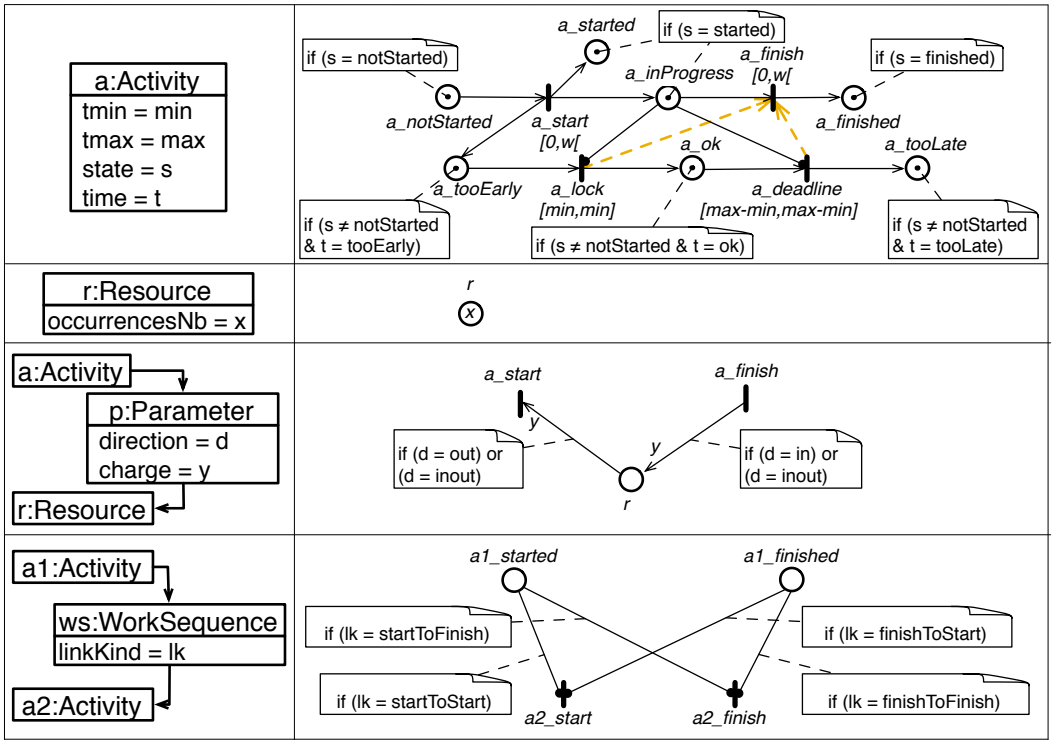

Figure 6: xSPEM2PetriNet (Simplified) Translation

the behavior of the modeled activity. We rely on the use of priorities among transitions to soundly deal with temporal constraints. As an example, the *a_deadline* transition is defined with a higher priority than the *a_finish* one (light grey in Figure 6).

Each *Resource* is represented by one place where the initial marking is initialized with its number of occurrences (*occurrencesNb*). Every activity *Parameter* is translated into one arc whose *weight* is initialized with a *charge*. This arc is linked to one activity transition according to the *direction*.

A *WorkSequence* becomes a *read-arc* from one place of the source activity to a transition of the target activity according to the *linkKind*.

The hierarchical decomposition of activities is represented in the form of scheduling constraints in the following manner: (A1 ♦— A2) = ((A1 $\xrightarrow{S2S}$ A2) & (A2 $\xrightarrow{F2F}$ A1)), $S2S$ denoting a *startToStart* dependency between activities $A1$ and $A2$ and $F2F$ a *finishToFinish* one (see *WorkSequenceType* on Figure 3).

States Mapping Finally, the process state is characterized using the markings of places characterizing the activity state and the local clock. The different alternatives are expressed in the Figure 6 through the use of annotations. These annotations are complete with respect to the possible combination of values for semantics variables $state$ and $time$.

4.4 Implementing xSPEM2PetriNet Transformation Through MDE Practices

In order to fit the MDE view, the target model must also be defined as a model as well as the translation from xSPEM to Petri nets.

We first propose a Petri net metamodel that fits the usual definition defined above in Section 4.2. Then we rely on this metamodel to define a model transformation from xSPEM metamodel to the Petri net one.

Petri Net Metamodel Figure 7 proposes a possible metamodel for PrTPN. A *PetriNet* is composed of *Nodes* that denote *Places* or *Transitions*. Nodes are linked by *Arcs*. Arcs can be normal ones or read-arcs. An *Arc* specifies the number of tokens consumed in the source place or produced in the target one (*weight*). A *read-arc* only checks tokens availability without removing them. A Petri net marking is defined by the number of tokens contained in each place (*marking*). Priorities are modeled as a self-reference on the *Transition* element, the source transition having a higher priority that the target one. Finally, a time interval can be expressed on a *Transition*. Obviously many models conforming to this metamodel are invalid models. As an ex-

ample, this metamodel does not prevent from putting an arc between two places or two transitions. Thus, we have completed it with OCL rules to check whether models are valid or not. The metamodel, embedding structural rules based on OCL rules and associated to its semantics, is our target DSL.

Model Transformation The transformation described in Section 4.3 and Figure 6 has been written in ATL. The complete sources are available in the TOPCASED[5] open source project. The principles of this approach are detailed through a complete case study on the Eclipse website within the context of an execution dedicated XSPEM subset (SIMPLEPDL)[6].

Remark 1 (Translation Π) *We denote by Π the function that applies the ATL transformation described above on a model. It is defined for every model that conforms to* XSPEM. *As mentionned earlier in Section 3.2.3, this function is later used to reason about the initial model and to guaranty the validity of the translation.*

The Petri net model is then translated into the concrete syntax of Tina, our target Petri net model checker, using an ATL query PETRINET2TINA. To target other Petri nets tools, only this last transformation would have do be adapted.

Now that the process model is translated into a Petri net model, we can check XSPEM properties by using TINA [5]. Properties expressed on the XSPEM metamodel are matched against an ATL transformation that produces the corresponding LTL properties instantiated from the XSPEM model.

There are two kinds of checked properties: universal or existential. In the first case, the property must be checked in all executions. If they fail, the tool provides a trace counter-example. The second case corresponds to checking that one possible execution satisfies the property, for example the time or resources constraints. If such an execution exists, its trace is generated by the tool. This kind of property is usually obtain with model-checking tools, by trying to ensure the validity of their negation. The counter example produced by the tool is the answer to the initial query.

[5] http://www.topcased.org

[6] http://eclipse.org/m2m/atl/usecases/SimplePDL2Tina

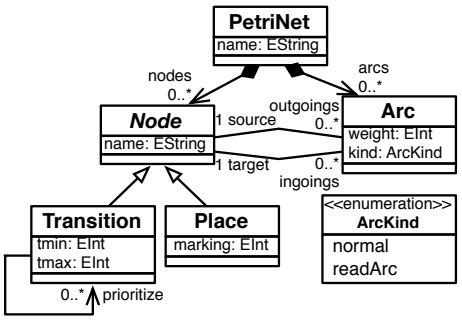

Figure 7: PRIORITIZED TIME PETRI NET Metamodel

4.5 Validating the translation

Finally, we establish a bisimulation relation between the two semantics. This relation ensures that the conclusions obtained on the Petri nets also hold on the XSPEM model: a property checked by the Tina Petri net model checker we used, is thus a valid property on XSPEM.

Let us first compare the number of possible transitions in the XSPEM model and in its associated Petri net. In the first one we have two transitions applicable to each activity whereas in the second one we have four transitions for each encoded activity. Therefore we need to prove a weak bisimulation between these two models.

Theorem 1 (Weak bisimulation) *Model state space MS and Prioritized Time Petri Nets state space PNS are in bisimulation w.r.t. the translation function Π, according to the definition 1.*

Proof 1 *The theorem is proved by induction on the process model structure. The initial case addresses the bisimilarity of a single activity and its encoding. Then by structural induction on the number of activities and their dependences, one prove the theorem. The property for a set of activities to be part of a bigger one is encoded by dependence links and thus is preserved by the bisimulation. The different steps of this weak bissimulation proof are detailed in appendix A.*

5 Related Work

Related works presented in the survey on the definition of semantics (Section 2) are not focused on the problem of consistency between several complementary semantics because they aim at defining one semantics for a given language. This is achieved either through operational semantics or translational semantics.

Translational semantics is often used to reuse available tools of target technical space like code generators, model-checkers and so one. The work of [20] is close to the application domain we have used in this paper. Indeed, the authors propose a specific mapping from BPMN (Business Process Modeling Notation) to Petri net in order to analyse business process models. Like most of related works, the stress is not on the check of the consistency of the

BPMN semantics and the one given by the translation into Petri nets. Furthermore, they do not propose a general and generic approach to describe a translational semantics according to an operational one.

The key point is that we believe that a semantics always exists on the DSL even if it is often only implicit or informally described. We advocate that this semantics, the reference semantics, has to be explicitly described so that it can be validated by the DSL designer. One step toward this goal is certainly achieved thanks to the work done to make the definition of semantics easier for example by providing a friendlier language to express it like visual graphical notation of graph transformation rules [21], Story Diagrams [23] or by reusing already defined semantics units like [12].

Furthermore, we can notice that aside the reference semantics of the DSL, several semantics may have to be defined for the same DSL to address the problem at different levels of abstraction or for reusing the tools available in other technical spaces. So we had to deal with the consistency of these different definitions of the semantics of the same DSL.

Automatically generating the mapping defining the translational semantics is one way that has been investigating to ensure this consistency. The approach proposed in [19] is one example. Once the semantics is defined in an operational way on the DSL (through graph rewriting rules), they are able to translate a model of this DSL into a Petri net having the same behavior *by construction*. This translation is based on the definition of a mapping between DSL metamodel and the Petri net metamodel expressed through *Triple Graph Grammar* (TGG). It consists in stating if an element of the source DSL becomes a place or a token in the target Petri net. Each rewriting rule describing the behavior of the DSL is automatically translated into a Petri net transition. Unfortunately, the approach imposes strong hypothesis and thus cannot translate arbitrary behavioral specifications. It also constrains the metamodel because a source element can only be map into one place or one token. In our case, the xSPEM metamodel should be changed to define one subclass of each possible state of an activity.

The same approach is used in [47]. As it has been presented in Section 2, they translate an operational semantics defined on the DSL into the Maude environment.

A second way to verify consistency of semantics has been proposed by Narayanan and Karsai [39]. As a first step towards verifying model transformations, Narayanan and Karsai propose to check if a particular generated model is a valid representation of a particular source model in order to verify a given property about the source model. They establish an equivalence relation between objects of the input and output models and use it to check if the two models are similar in behavior. The approach has been applied to a transformation from statecharts to EHA (Extended Hybrid Automata) and the checking is done according to links between input and output objects recorded during the transformation execution. In [40] they use semantic anchoring [12] to the same semantic unit (Finite State Machines, FSM defined using Abstract State Machines, ASM) to define the semantics of two variants of statechart, and then check a weak bisimulation between the two resulting FSM models to ensure that they are behaviorally equivalent.

The approach does not ensure that both semantics are consistent but it can assert whether the target model is behaviorally equivalent to the source one for a given property. The check has to be done for each transformation but as it is included in the transformation process, its execution is automatic.

The work is facilitated by the fact that the two metamodels are quite similar and the transformation produces a one to one mapping for states and transitions of both metamodels. It seems to be far less obvious in the case of general metamodels, for example in the case of xSPEM to Petri nets. In order to help detect errors in the transformation itself, we have defined additional LTL properties that controls that invariants on the source metamodel are preserved of the target metamodel. For example, we can check that the same activity cannot be at the same time not started, running or finished. Obviously, it does not ensure that the transformation is correct and only helps in pointing out errors.

In the process of validating model transformations, the notion of bisimulation is a central concern. Actually, many different definitions of bisimulation do exist, depending upon the chosen granularity between corresponding events of the two systems to be proved bisimilar. These bisimulations have been defined on a semantical basis, as relations between transition systems, but for a vast majority of works, they have been applied to process calculi, and especially π-calculus [52]. Still, the focus is mainly being put on defining new variants and addressing their properties, disregarding the eventuality of automating bisimulation proofs.

As for this last topic, some works about automatic proofs of bisimulations recently came up, again for π-terms and within the framework of a proof assistant: Coq [54]. As is, the results presented in [26, 46] seem inapplicable to our case without a considerable redesign effort. Indeed, their heuristics for automating bisimulation proofs have been specifically developed for π-terms, not for arbitrary transition systems, and are obviously not aware of our metamodeling framework. As far as we know, the amount of related works seems quite small. And furthermore, the special case when one system is obtained through translation from the other is by far an unexplored territory, even for structural and modular translations like ours.

To conclude with a positive remark, most theoretical as well as practical results about automation of bisimulation proofs stemed from the study of finite state systems, a class

the simple xSPEM models presented in this paper indeed boil down to. In this context, many questions about bisimulations are decidable, though usually very resource demanding, as advocated by a bunch of related tools [3, 8].Yet this appealing property doesn't carry over to the general case of xSPEM models.

6 Conclusion and Perspectives

This paper gives several contributions on the definition of model semantics. A first one is a survey of the different techniques allowing to define and manipulate the execution of models. The different approaches are compared with their pros and cons. Then a second contribution is the application of one approach, the translational semantics definition, on a DSL describing processes. Its semantics is defined through a mapping to prioritized time Petri nets. All the step towards the sound definition of the DSL semantics are detailed. We formalize the initial semantics, give the semantics of the target DSL, provide a translation from the source metamodel to Petri nets. This transformation is validated by a bisimulation proof.

This work is a first step toward the practical instrumentation of models. In particular the approach presented here, including the transformation validation step, ensures that one can rely on all available tools on the target model while keeping a strong confidence in the quality of the semantics representation. For example, model can then be validated using static analyzers, model checkers, or even be simulated using specific tools.

It is now essential to continue this work with the interpretation of the results obtained on the Petri nets in term of the xSPEM concepts. It will then provide a transparent way to instrument high level models and gives tool to non expert in order to manipulate their models. For example, a trace obtained by a model checker on the target model could be translated back as a trace of the source model (in xSPEM) and exploited by an xSPEM simulator. Traces corresponding to counterexamples may be used to find errors on the source model while traces of an existential property may be used to simulate a possible execution.

Another perspective is a computer-aided way to build such translational semantics. The bisimulation proof step which seems necessary could be automated, at least partially. Promising works of [26] and [46] address the automated verification of bisimulation, resp. strong and weak.

In this line of thought, we have started the design and implementation of a formal framework for expressing formal semantics of models, the Coq4MDE (Coq for Model-Driven Engineering) framework, which principles are exposed in [55]. The rationale behind Coq4MDE was to provide a formal foundation to the various concepts of MDE (e.g. model, metamodel, model conformity, model transformation, etc) so that properties could be stated and proved about general MDE concepts and some of their specific instances. We make use of the general purpose higher-order logic and proof assistant Coq, as it provides an automatic mechanism for generating executable programs from proofs, loosely speaking. Thus, an operational formal reference semantics in this framework could be automatically turned into an executable semantics, suitable for testing and interactive simulation for instance, with no supplementary developing effort. An immediate benefit of this executable semantics is the possibility to put the model designers' reference semantics to a test, in early stages of DSL definition, i.e., before attempting to define a translational semantics with its bisimulation proof. Such a feedback would surely help in designing a sensible and suitable reference semantics in a more efficient way.

As a general conclusion, regarding the spread of model driven engineering, more and more tools will be needed to support model manipulation, in particular model execution. This work proposes an approach allowing to rely on existing formal models and tools to support new developments.

References

[1] A. Agrawal, G. Karsai, Z. Kalmar, S. Neema, F. Shi, and A. Vizhanyo. The Design of a Language for Model Transformations. Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA, 2005.

[2] R. Bendraou, B. Combemale, X. Crégut, and M.-P. Gervais. Definition of an eXecutable SPEM2.0. In *14th APSEC*, Japan, Dec. 2007. IEEE Computer Society.

[3] D. Bergamini, N. Descoubes, C. Joubert, and R. Mateescu. Bisimulator: A modular tool for on-the-fly equivalence checking. In N. Halbwachs and L. D. Zuck, editors, *TACAS*, volume 3440 of *LNCS*, pages 581–585. Springer, 2005.

[4] B. Berthomieu, F. Peres, and F. Vernadat. Model checking bounded prioritized time petri nets. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *ATVA*, volume 4762 of *LNCS*, pages 523–532. Springer, 2007.

[5] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *Int. Journal of Production Research*, 42(14):2741–2756, 2004.

[6] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. Modeling in the large and modeling in the small. In U. Aßmann, M. Aksit, and A. Rensink, editors, *Model Driven Architecture, European MDA Workshops: Foundations and Applications (MDAFA 2004)*, number 3599 in LNCS, pages 33–46, Linköping, Sweden, June 2004. Springer Verlag.

[7] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, and E. Weiss. Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Proceedings of the 9th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, vol-

ume 4199 of *Lecture Notes in Computer Science*, pages 425–439. Springer, Oct. 2006.

[8] A. Bouali. XEVE, an ESTEREL Verification Environment. In A. J. Hu and M. Y. Vardi, editors, *CAV*, volume 1427 of *LNCS*, pages 500–504. Springer, 1998.

[9] E. Breton. *Contribution à la représentation de processus par des techniques de méta-modélisation*. PhD thesis, Université de Nantes, June 2002.

[10] F. Budinsky, E. Merks, and D. Steinberg. *Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2009.

[11] F. Budinsky, D. Steinberg, and R. Ellersick. *Eclipse Modeling Framework : A Developer's Guide*. Addison-Wesley Professional, 2003.

[12] K. Chen, J. Sztipanovits, S. Abdelwalhed, and E. Jackson. Semantic anchoring with model transformations. In *Model Driven Architecture - Foundations and Applications, First European Conference (ECMDA-FA)*, volume 3748 of *LNCS*, pages 115–129, 2005.

[13] T. Clark, A. Evans, and S. Kent. The Metamodelling Language Calculus: Foundation Semantics for UML. In *Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 2029 of *Lecture Notes In Computer Science*, pages 17–31, London, UK, 2001. Springer.

[14] T. Clark, A. Evans, P. Sammut, and J. Willans. Applied metamodelling - a foundation for language driven development. version 0.1, 2004.

[15] T. Clark, P. Sammut, and J. Willans. Applied Metamodelling – A Foundation for Language Driven Development. Second Edition, 2008.

[16] T. Clark, P. Sammut, and J. Willans. SUPERLANGUAGES – Developing Languages and Applications with XMF. First Edition, 2008.

[17] T. Cleenewerck and I. Kurtev. Separation of concerns in translational semantics for DSLs in model engineering. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*, pages 985–992, New York, NY, USA, 2007. ACM Press.

[18] B. Combemale, S. Rougemaille, X. Crégut, F. Migeon, M. Pantel, C. Maurel, and B. Coulette. Towards a Rigorous Metamodeling. In *2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS)*, Paphos, Cyprus, May 2006. INSTICC.

[19] J. de Lara and H. Vangheluwe. Translating model simulators to analysis models. In J. Fiadeiro and P. Inverardi, editors, *11th International Conference Fundamental Approaches to Software Engineering (FASE)*, volume 4961 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2008.

[20] R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008.

[21] G. Engels, R. Heckel, and S. Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In *UML 2000 - The Unified Modeling Language. Advancing the Standard, vol. 1939 of LNCS*, pages 323–337. Springer, 2000.

[22] P. Farail, P. Gaufillet, A. Canals, C. L. Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel. The TOPCASED project: a Toolkit in OPen source for Critical Aeronautic SystEms Design. In *Embedded Real Time Software (ERTS'06)*, Toulouse, 25-27 January 2006.

[23] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph transformation language based on UML and Java. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Theory and Application to Graph Transformations (TAGT'98), Paderborn, November, 1998*, volume 1764 of *LNCS*. Springer, 1998.

[24] M. Gogolla, P. Ziemann, and S. Kuske. Towards an Integrated Graph Based Semantics for UML. In P. Bottoni and M. Minas, editors, *Proceedings of the Graph Transformation and Visual Modeling Techniques (GT-VMT)*, volume 72(3) of *ENTCS*, Barcelona, Spain, Oct. 2002. Elsevier.

[25] J. H. Hausmann. *Dynamic Meta Modeling – A Semantics Description Technique for Visual Modeling Languages*. PhD thesis, University of Paderborn, 2005.

[26] D. Hirschkoff. Bisimulation Verification Using the Up-to Techniques. *STTT*, 3(3):271–285, 2001.

[27] F. Jouault and J. Bézivin. KM3: a DSL for Metamodel Specification. In *IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 4037 of *LNCS*, pages 171–185. Springer, 2006.

[28] F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS*, LNCS, Jamaica, 2005. Springer.

[29] H. Kastenberg, A. Kleppe, and A. Rensink. Defining Object-Oriented Execution Semantics Using Graph Transformations. In R. Gorrieri and H. Wehrheim, editors, *Proceedings of the 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, volume 4037 of *Lecture Notes in Computer Science*, pages 186–201, Bologna, Italy, June 2006. Springer-Verlag.

[30] H. Kastenberg, A. Kleppe, and A. Rensink. Engineering Object-Oriented Semantics Using Graph Transformations. CTIT Technical Report 06-12, University of Twente, March 2006.

[31] S. Kuske. *Transformation Units—A structuring Principle for Graph Transformation Systems*. PhD thesis, University of Bremen, 2000.

[32] S. Kuske. A Formal Semantics of UML State Machines Based on Structured Graph Transformation. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, volume 2185 of *Lecture Notes In Computer Science*, pages 241–256, London, UK, 2001. Springer.

[33] S. Kuske, M. Gogolla, R. Kollmann, and H.-J. Kreowski. An Integrated Semantics for UML Class, Object and State Diagrams Based on Graph Transformation. In *Proceedings of the 3rd International Conference on Integrated Formal Methods (IFM)*, volume 2335 of *Lecture Notes In Computer Science*, pages 11–28, London, UK, 2002. Springer.

[34] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. T. IV, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing*, Budapest, Hungary, 2001.

[35] S. Markovic and T. Baar. Semantics of OCL specified with QVT. *Software and System Modeling*, 7(4):399–422, 2008.

[36] P. M. Merlin. *A Study of the Recoverability of Computing Systems*. Irvine: Univ. California, PhD Thesis, 1974.

[37] R. Milner. *Communication and concurrency*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, c. a. r. hoare edition, 1995.

[38] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. Weaving executability into object-oriented meta-languages. In S. K. L. Briand, editor, *MoDELS*, LNCS, Jamaica, 2005. Springer.
[39] A. Narayanan and G. Karsai. Towards verifying model transformations. In R. Bruni and D. Varró, editors, *5th International Workshop on Graph Transformation and Visual Modeling Techniques, Vienna*, pages 185–194, Apr. 2006.
[40] A. Narayanan and G. Karsai. Using semantic anchoring to verify behavior preservation in graph transformations. *ECEASST*, 4, 2006.
[41] Object Management Group, Inc. *UML Object Constraint Language (OCL) 2.0 Specification*, Oct. 2003. Final Adopted Specification.
[42] Object Management Group, Inc. *Meta Object Facility (MOF) 2.0 Core Specification*, Jan. 2006. Final Adopted Specification.
[43] Object Management Group, Inc. *Software Process Engineering Metamodel (SPEM) 2.0*, Mar. 2007.
[44] Object Management Group, Inc. *Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.0*, Apr. 2008.
[45] R. F. Paige, D. S. Kolovos, and F. A. C. Polack. An action semantics for MOF 2.0. In *Proceedings of the 2006 ACM symposium on Applied computing (SAC)*, pages 1304–1305, New York, NY, USA, 2006. ACM.
[46] D. Pous. New up-to techniques for weak bisimulation. *Theor. Comput. Sci.*, 380(1-2):164–180, 2007.
[47] J. E. Rivera, E. G. annd Juan de Lara, and Antonio. Analyzing Rule-Based Behavioral Semantics of Visual Modeling Languages with Maude. In *International Conference on Software Language Engineering*, Oct. 2008.
[48] J. E. Rivera, J. R. Romero, and A. Vallecillo. Behavior, time and viewpoint consistency: Three challenges for mde. In *Proc. of the First International Workshop on Challenges in Model-Driven Software Engineering (ChaMDE'2008)*, LNCS. Springer, 2008.
[49] J. E. Rivera and A. Vallecillo. Adding behavioral semantics to models. In *11th IEEE International Enterprise Distibuted Object Computing Conference. EDOC 2007, 15-19 October 2007 Annapolis, Maryland, USA. Proceedings*, pages 169–180, Los Alamitos, California, Oct. 2007. IEEE Computer Society.
[50] J. R. Romero, J. E. Rivera, F. Duran, and A. Vallecillo. Formal and Tool Support for Model Driven Engineering with Maude. *Journal of Object Technology, Special Issue: TOOLS EUROPE 2007*, 6(9):187–207, Oct. 2007.
[51] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.
[52] D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inf.*, 33(1):69–97, 1996.
[53] G. Taentzer. AGG : A Graph Transformation Environment for Modeling and Validation of Software. In Springer-Verlag, editor, *AGTIVE*, volume 3062 of *LNCS*, pages 446–453, 2003.
[54] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.1*, 2006. http://coq.inria.fr.
[55] X. Thirioux, B. Combemale, X. Crégut, and P.-L. Garoche. A Framework to formalise the MDE Foundations. In R. Paige and J. Bézivin, editors, *Proceedings of the International Workshop on Towers of Models (TOWERS)*, pages 14–30, Zurich, June 2007.
[56] G. Winskel. *The formal semantics of programming languages: an introduction*. MIT Press, Cambridge, MA, USA, 1993.
[57] P. Ziemann, K. Hölscher, and M. Gogolla. From UML Models to Graph Transformation Systems. In M. Minas, editor, *Proceedings of the Workshop on Visual Languages and Formal Methods (VLFM)*, volume 127(4) of *ENTCS*. Elsevier, 2005.

A Proofs

In this appendix, we give more details about the different steps of the weak bisimulation proof.

Lemma 1 *Model state space MSS where states are restricted to a single activity and Prioritized Time Petri Nets state space PNS are in bisimulation w.r.t. the translation function Π, according to the definition 1.*

Proof 2 *Let S be a model state and $u \in T*$ be such that $S_0 \xrightarrow{u} S$,*

1. *Let S' be a model states containing only a single Activity such that $S \xrightarrow{\lambda} S'$.*

   Then $\Pi(S)$ is the Petri net described in Fig 6 describing only one single activity in PN.

$$\begin{array}{rcl} I_s & = & \left\{ \begin{array}{l} s \mapsto (0,0), f \mapsto (0,w), l \mapsto (min, min), \\ d \mapsto (max - min, max - min) \end{array} \right. \\ P & = & \{a\_notStarted, a\_started, a\_inProgress, \\ & & a\_finished\} \cup \{a\_tooEarly, a\_ok, a\_tooLate\} \\ T & = & \{a\_start, a\_finish\} \cup \{a\_lock, a\_deadline\} \end{array}$$

   We associate to the transitions $StartActivity$ and $FinishActivity$ of xSPEM *semantics the Petri net transitions a_start and a_finish respectively. The two Petri nets remaining transitions a_lock and $a\_deadline$ are our epsilon transitions, the transitions that are not observable.*

   We now consider the different cases of possible transitions applicable on S:

   - *$StartActivity$ with any clock value ($\forall \theta$)*

     In that case, the activity in S is such that $\exists clock, (notStarted, ok, clock)$. The precondition about predecessor is satisfies since the considered activity is the only one in the state.

     Then S is such that the marking obtained by Π contains a single token in the place nS.

     $$m = \{a\_notStarted \mapsto 1\}, I = \{a\_start \mapsto (0, w)\}$$

     According to the semantics of PN, $m > Pre(s)$. Let us compute such transition, the resulting Petri net (m', I') is such that:

$$\begin{array}{rcl} m' & = & \{a\_started \mapsto 1, a\_inProgress \mapsto 1, \\ & & a\_tooEarly \mapsto 1\}, \\ I' & = & \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\}. \end{array}$$

Epsilon transitions are not computable here.

Let us go back to S'. According to the semantics of MS, S' is our activity with values $(started, ok, 0)$. Its image by the Π function gives a Petri net (m'', I'') such that

$$\begin{aligned} m'' &= \{a\_started \mapsto 1, a\_inProgress \mapsto 1, \\ &\quad a\_tooEarly \mapsto 1\}, \\ I'' &= \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\}. \end{aligned}$$

We have $(m', I') = (m'', I'')$

- *$FinishActivity$ with $\theta < min$*

  Let us consider the second case. S is such that its state is described by the triple $\exists clock$, such that $(started, ok, clock)$ and $clock < min\_time$.

  The resulting S' is such that its state is described by the triple $(finished, tooEarly, clock)$.

  The image of S by Π is (m, I) with

$$\begin{aligned} m &= \{a\_started \mapsto 1, a\_inProgress \mapsto 1, \\ &\quad a\_tooEarly \mapsto 1\}, \\ I &= \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\} \end{aligned}$$

  $\Pi(S')$ is defined as (m', I') where

$$\begin{aligned} m' &= \{a\_started \mapsto 1, a\_finished \mapsto 1, \\ &\quad a\_tooEarly \mapsto 1\}, \\ I &= \{\} \end{aligned}$$

  Let us show that

$$(m, I) \xrightarrow{(\tau,\theta_1)*} \xrightarrow{(f,\theta_2)} \xrightarrow{(\tau,\theta_3)*} (m'', I'')$$

  with $\theta_1 + \theta_2 + \theta_3 = \theta = clock$

  The first transition on $a\_lock \in \tau$ is not applicable, since $\theta_1 < min$.

$$(m, I) \xrightarrow{(f,\theta_2)} (m''', I''')$$

  with $m''' = \{a\_started \mapsto 1, a\_tooEarly \mapsto 1, a\_finished \mapsto 1\}, I''' = \{\}$ Then no τ transition is applicable.

  And we have $(m''', I''') = (m'', I'') = (m, I)$

- *$FinishActivity$ with $min \leq \theta < max$*

  $S = (started, ok, clock)$ and $min\_time \leq clock < max\_time$.

  $S' = (finished, ok, clock)$

  $\Pi(S) = (m, I)$ with

$$\begin{aligned} m &= \{a\_started \mapsto 1, a\_inProgress \mapsto 1, \\ &\quad a\_tooEarly \mapsto 1\}, \\ I &= \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\} \end{aligned}$$

  $\Pi(S') = (m', I')$ with

  $m' = \{a\_started \mapsto 1, a\_finished \mapsto 1, a\_ok \mapsto 1\}, I = \{\}$

  Let us now show that

$$(m, I) \xrightarrow{(\tau,\theta_1)*} \xrightarrow{(f,\theta_2)} \xrightarrow{(\tau,\theta_1)*} (m', I')$$

  with $\theta_1 + \theta_2 + \theta_3 = \theta = clock$

  Transitions a_lock and a_finish are applicable to (m, I).

  The transition $a\_lock \in \tau$ could be applicable since $m > Pre(l)$. If $\theta_1 < min$ then the transition a_lock is not applicable. A first case is when $(m, I) \xrightarrow{(f,\theta_2)} (m_2, I_2)$ with $m_2 = \{a\_started \mapsto 1, a\_finished \mapsto 1, a\_tooEarly \mapsto 1\}$, $I_2 = \{\}$ and $\theta_2 = clock$ But according to the PN semantics, θ_2 must then be $< min$. The transition is not computable.

  Then θ_1 must be $\geq min$. Furthermore $\theta_1 \leq w$. The transition occurs.

$$(m, I) \xrightarrow{(l,\theta_1)} (m'_2, I'_2)$$

  with $m'_2 = \{a\_started \mapsto 1, a\_ok \mapsto 1, a\_inProgress \mapsto 1\}, I'_2 = \{a\_deadline \mapsto (max - min, max - min), a\_finish \mapsto (0, w)\}$ (we have $m - Pre(l) < Pre(d)$ & $m - Pre(l) < Pre(f)$)

  We have $0 \leq \theta_2 + \theta_3 < max - min$. Let us now compute the transition f.

$$(m'_2, I'_2) \xrightarrow{(f,\theta_2)} o(m_3, I_3)$$

  with $m_3 = \{a\_started \mapsto 1, a\_ok \mapsto 1, a\_finish \mapsto 1\} and I_3 = \{\}$

  We obtain $(m', I') = (m_3, i_3)$

- *$FinishActivity$ with $\theta > max$*

  $S = (started, ok, clock)$ and $clock > max\_time$.

  $S' = (finished, tooLate, clock)$

  $\Pi(S) = (m, I)$ with

$$\begin{aligned} m &= \{a\_started \mapsto 1, a\_inProgress \mapsto 1, \\ &\quad a\_tooEarly \mapsto 1\}, \\ I &= \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\} \end{aligned}$$

  $\Pi(S') = (m', I')$ with

$$\begin{aligned} m' &= \{a\_started \mapsto 1, a\_finish \mapsto 1, \\ &\quad a\_tooLate \mapsto 1\}, \\ I &= \{\} \end{aligned}$$

  Let us now show that

$$(m, I) \xrightarrow{(\tau,\theta_1)*} \xrightarrow{(f,\theta_2)} \xrightarrow{(\tau,\theta_3)*} (m', I')$$

  with $\theta_1 + \theta_2 + \theta_3 = \theta = clock$

  Transitions a_lock and a_finish are applicable to (m, I).

  The same reasoning to the last case applies here. Then necessary, a first a_lock transition occurs when $\theta_1 \geq min$ and $\theta_1 < w$.

$$(m, I) \xrightarrow{(a\_lock,\theta_1)} (m'_2, I'_2)$$

with

$$m'_2 = \{a\_started \mapsto 1, a\_ok \mapsto 1, a\_inProgress \mapsto 1\},$$
$$I'_2 = \left\{ \begin{array}{l} a\_deadline \mapsto (max - min, max - min), \\ a\_finish \mapsto (0, w) \end{array} \right\}$$

$(m - Pre(l) < Pre(d)$ & $m - Pre(l) < Pre(f))$
Let's see if the transition on f can apply. Then $\theta_2 < max - min$. And no more transition could apply. But $clock = \theta_1 + \theta_2 = max$ and we consider the case $clock > max$
Then we have to compute the transition on d $(m'_2, I'_2) \xrightarrow{(a\_deadline, max-min)} (m''_2, I''_2)$ *with*

$$m''_2 = \{a\_started \mapsto 1, a\_tooLate \mapsto 1, a\_inProgress \mapsto 1\},$$
$$I''_2 = \{a\_finish \mapsto (0, w)\}$$

Then transition on f can then apply. $(m''_2, I''_2) \xrightarrow{(a\_finish, \theta_2)} (m_3, I_3)$ *with* $m_3 = \{a\_started \mapsto 1, a\_tooLate \mapsto 1, a\_finish \mapsto 1\}$ *and* $I_3 = \{\}$
We obtain $(m', I') = (m_3, i_3)$

2. *Let P' be a Petri net state such that* $\Pi(S) \xrightarrow{\lambda} P'$.

- $S = (notStarted, notFinished, clock)$

  $(m, I) = (\{a\_notStarted \mapsto 1\}, \{a\_start \mapsto (0, w)\})$

  There is only one possible transition $\exists\theta$ *s.t.* $(m, I) \xrightarrow{(a\_start, \theta)} (m', I')$

  $$m' = \{a\_start \mapsto 1, s\_inProgress \mapsto 1, s\_tooEarly \mapsto 1\}$$
  $$I' = \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\}$$

  The image of S by the same transition gives $S' = (started, notFinished, 0)$
  and $\Pi(S') = (m', I')$
- $S = (started, notFinished, clock)$
  $\Pi(S) = (m, i)$ *with*

  $$m = \{a\_started \mapsto 1, a\_inProgress \mapsto 1, a\_tooEarly \mapsto 1\},$$
  $$I = \{a\_lock \mapsto (min, min), a\_finish \mapsto (0, w)\}$$

  We have two possibilities :
  - *applying* a_finish *iff* $\theta < min$ *then necessary,* $(m, I) \xrightarrow{(f, \theta)} (\{a\_started \mapsto 1, a\_finished \mapsto 1, a\_tooEarly \mapsto 1\}, \{\}) = (m', I')$ *and* $clock = \theta < min$ $S \xrightarrow{\lambda_2} (finished, tooEarly, clock)$ $\Pi((finished, tooEarly, clock)) = (m', I')$
  - *applying* a_lock *iff* $\theta_1 = min$ $(m, i) \xrightarrow{(a\_lock, \theta_1)} (\{a\_started \mapsto 1, a\_inProgress \mapsto 1, a\_ok \mapsto 1\}, \{a\_deadline \mapsto (max - min, max - min), a\_finish \mapsto (0, w)\}) = (m_2, I_2)$ *We have now two cases again:*
    - *(a)* $a\_deadline$ *iff* $\theta_2 = max - min$ *then* $(m_2, I_2) \xrightarrow{(a\_deadline, \theta_2)} (\{a\_started \mapsto 1, a\_tooLate \mapsto 1, a\_inProgress \mapsto 1\}, \{a\_finish \mapsto (0, w)\}) = (m_3, I_3)$
      Finally the transition f can apply. $(m_3, I_3) \xrightarrow{(a\_finish, \theta_3)} (\{a\_started \mapsto 1, a\_tooLate \mapsto 1, a\_finished \mapsto 1\}, \{\}) = (m', I')$
      and $clock = \theta_1 + \theta_2 + \theta_3 = max + \theta_3$ $S \xrightarrow{\lambda_4} (finished, tooLate, clock)$ $\Pi((finished, tooLate, clock)) = (m', I')$
    - *(b)* a_finish *iff* $\theta_2 < max - min$ *then* $(m_2, I_2) \xrightarrow{(a\_finish, \theta_2)} (\{a\_started \mapsto 1, a\_ok \mapsto 1, a\_finish \mapsto 1\}, \{\}) = (m', I')$ $clock = \theta_1 + \theta_2 < max \& clock \geq min$ $S \xrightarrow{\lambda_3} (finished, ok, clock)$ $\Pi((finished, ok, clock)) = (m', I')$
  - *other cases of values for S are mapped to Petri net with not applicable transitions*

3. *Initial case. Trivially* $(m, I) = \Pi(S_0)$ *is defined and satisfied the property.*

Lemma 2 *Let us consider a process model state $S \in MS$ with a finite number n of activities with dependence rules among them such that S and $\Pi(S)$ are weakly bisimilar. Let us define the process model state $S' \supseteq S \in MS$ defined as the process model state S with one more activity A with no links. Then S' and $\Pi(S')$ are weakly bisimilar.*

Proof 3 *If no link exists between S and A in S' then*

- $S \rightarrow X \implies S \cup A \rightarrow X \cup A$
- *Similarly in Petri net, since no dependency link exists between A and S then* $\Pi(S') = \Pi(S) \cup \Pi(A)$ *and* $\Pi(S) \rightarrow \Pi(X) \implies \Pi(S \cup A) = \Pi(S) \cup \Pi(A) \rightarrow \Pi(X) \cup \Pi(A)$. *The transition does not add links then* $\Pi(X) \cup \Pi(A) = \Pi(X \cup A)$.

A similar reasoning applies to transition on A in presence of S with no link between A and S.

- $A \rightarrow A' \implies S \cup A \rightarrow S \cup A'$
- $\Pi(A) \rightarrow \Pi(A') \implies \Pi(S \cup A) = \Pi(S) \cup \Pi(A) \rightarrow \Pi(S) \cup \Pi(A') = \Pi(S \cup A')$.

Since S and $\Pi(S)$ are weakly bisimilar (by induction hypothesis) and using the lemma 1:

- *if a transition λ occurs on $S \subseteq S'$ then* $S \xrightarrow{\lambda} X \implies \Pi(S') \xrightarrow{\lambda} \Pi(X \cup A)$;
- *if a transition λ occurs on $A \subseteq S'$ then* $A \xrightarrow{\lambda} A' \implies \Pi(S') \xrightarrow{\lambda} \Pi(S \cup A')$;
- *if a transition λ occurs on $\Pi(S) \subseteq \Pi(S')$ then* $\Pi(S) \rightarrow \Pi(X) \implies \Pi(S') \xrightarrow{\lambda} \Pi(X \cup A)$
- *if a transition λ occurs on $\Pi(A) \subseteq \Pi(S')$ then* $\Pi(A) \rightarrow \Pi(A') \implies \Pi(S') \xrightarrow{\lambda} \Pi(S \cup A')$

Then S' and $\Pi(S')$ are in weak bisimulation.

Lemma 3 *Let us consider a process model state $S \in MS$ with a finite number n of activities with dependence rules among them such that S and $\Pi(S)$ are weakly bisimilar. Let us define the process model state $S' \supseteq S \in MS$ defined as the process model state S with one dependence link between two activity A_1 and $A_2 \in S$. Then S' and $\Pi(S')$ are weakly bisimilar.*

Proof 4 *The new dependence link constraints an activity A_2 by another one A_1. For all transition λ applicable to any activity $A \in S \setminus A_2$, the transition can occur in $\Pi(S')$ since S and $\Pi(S)$ are weakly bisimilar and the activity A is not constrained by the new link. And reciprocally, if the transition can occurs in $\Pi(A) \subseteq \Pi(S)$ then it can occur in $\Pi(S')$.*

Let us now consider the transitions applicable on A_2 in S' depending on the new link added. We can add as a preliminary remark that if a transition can occur on A_2 in S', it is also computable in S since the new dependence link does not exists there.

- *a link of type $start2start$*

  Then according to the definition of Fig. 4, all links targeting this activity A_2 and labeled start2start, resp. finish2start, must have their source activity in a started state, resp. finished state, in order to compute the start transition on A_2.

  Let us show that if this transition is computable in S' then it is in $\Pi(S')$.

  If the transition is computable in S', then all above constraint links in S constraining A_2 start satisfy their own constraints (either started or finished). $\Pi(S)$ is such that for each of these links there exists a read-arc in the resulting Petri net from $ax\_started$ or $ax\_finished$, depending on the link type, to $a2\_start$. Each of these read-arc source is a place fulfilled with a token (cf. preliminary remark).

  Furthermore, in $\Pi(S')$, the new link from A_1 to A_2 is also mapped to a read-arc from the place $a1\_started$ to transition $a2\_start$. The transition is computable in S' then the activity A_1 must be started. If so, its $a1\_started$ place has one token.

  The transition can then occurs in $\Pi(S')$.

- *a similar reasoning applies for finish2start, start2finish and finish2finish links.*

Reciprocally, in Petri nets,

- *image of a link of type $start2start$*

  $\Pi(S') = \Pi(S) \cup \{$ a new read-arc from $a1\_started$ to $a2\_started\}$. Then if $\Pi(S') \rightarrow Y$ using $a2\_start$ transition, then there must be at least one token in each place linked to $a2\_start$ by either an arc or a read-arc. Then by definition of Π and using the preliminary remark, S' is such that all activities constraining $A2$ start satisfy their own constraint including the new link. Then the transition can also occurs in S'.

- *a similar reasoning applies for finish2start, start2finish and finish2finish links.*

Since S and $\Pi(S)$ are weakly bisimilar (by induction hypothesis), we have S' and $\Pi(S')$ also weakly bisimilar.

Theorem 4 (Weak bisimulation) *Model state space MS and Prioritized Time Petri Nets state space PNS are in bisimulation w.r.t. the translation function Π, according to the definition 1.*

Proof 5 *By induction on the process model structure:*

- *The initial case is proved thanks to Lemma 1;*
- *Adding one activity preserves the property (Lemma 2);*
- *Adding one dependence link preserves the property (Lemma 3);*

The property for a set of activities to be part of a bigger one is encoded by dependence links and thus is preserved by the bisimulation.